Screens on Shuffle: Netflix's journey of SDUI Schema

Sreekanth Ramakrishnan

One Product. Many Inconsistent APIs.





BROWSE

BrowseAPI

`{rows}`

SEARCH

SearchAPI

`{ results, suggestions }`

NEW & HOT

NewHotAPI

`{ tabs, videos }`

POSTPLAY

PostPlayAPI

`{ postPlayItems }`

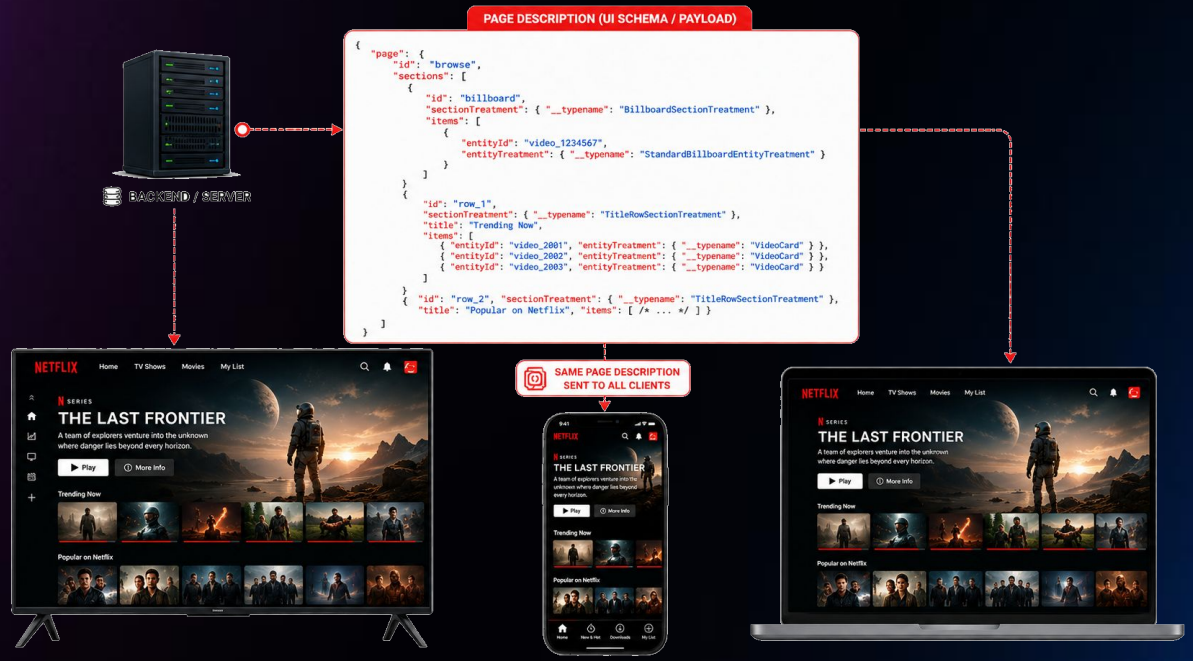
GAMES

GamesAPI

`{ gamesSections }`

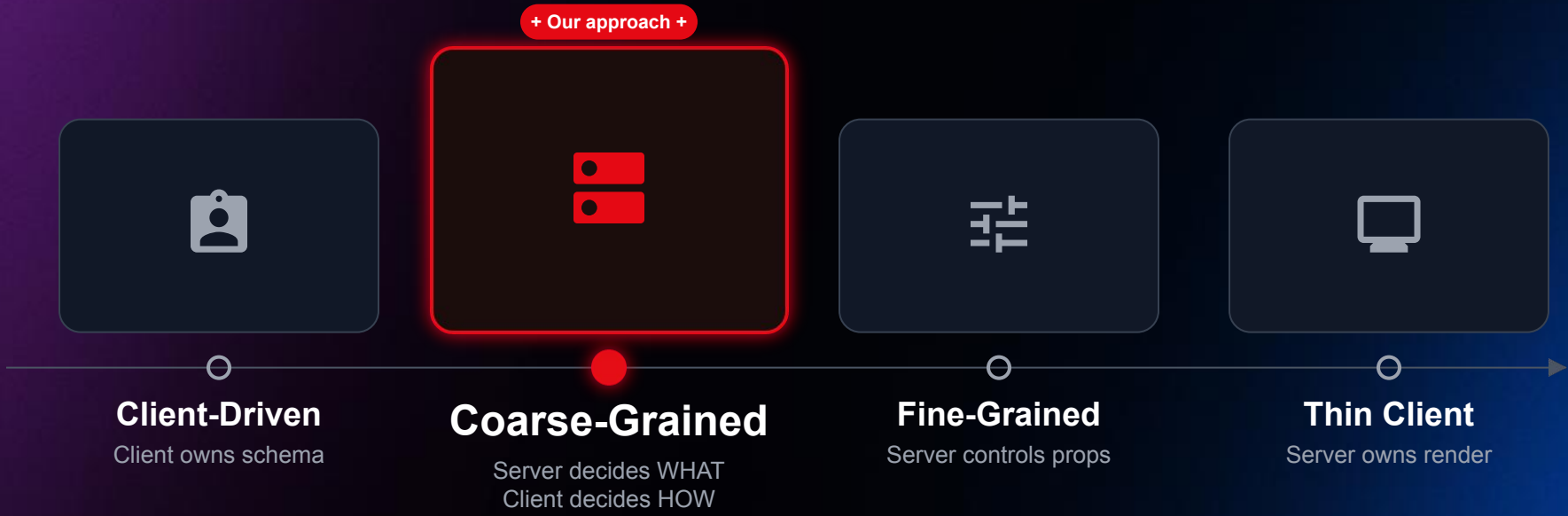
SDUI PHILOSOPHY

Server-Driven UI: Server describes the page. Client renders it.



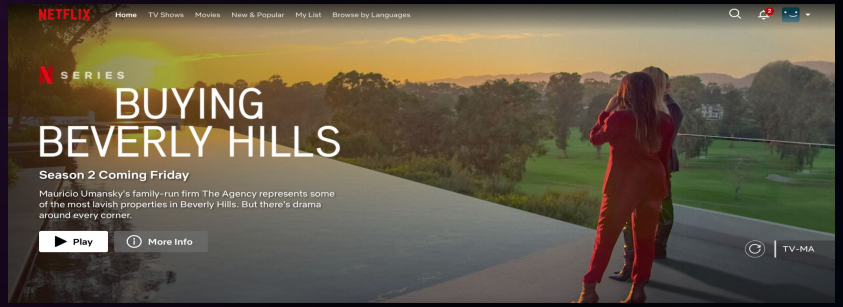
SDUI PHILOSOPHY

Coarse-Grained SDUI: Our Approach



SDUI PHILOSOPHY

The Same Movie. Four Different APIs.



CLIENT-DRIVEN

```
"id": "80189025"
```

Client queries every field.

+ Our approach +

COARSE-GRAINED

```
"entity": { "id": "80189025" },
"treatment":
"BillboardEntityTreatment"
```

Server sends a signal.

FINE-GRAINED

```
"entity": { "id": "80189025" },
"layout": {
"titleSize": 48,
"padding": "24px 32px",
"ctaStyle": "PRIMARY"
```

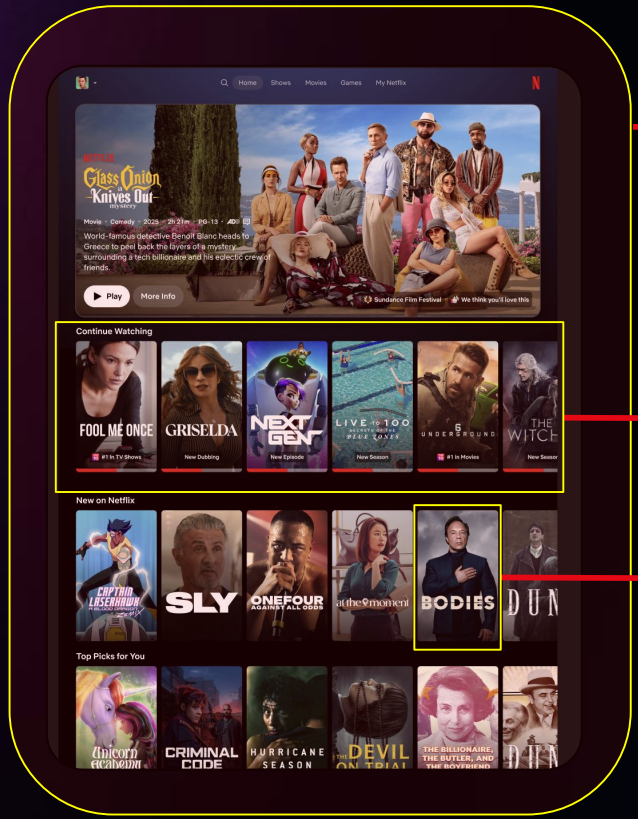
Server controls visual props and layout.

THIN CLIENT

```
<HTML>
...
</HTML>
```

.

Netflix Page Structure



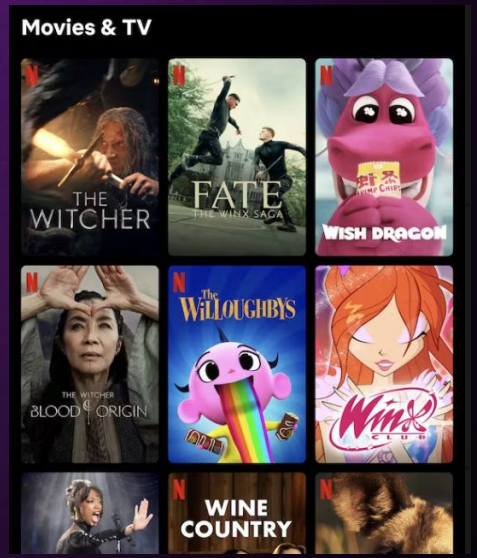
Page

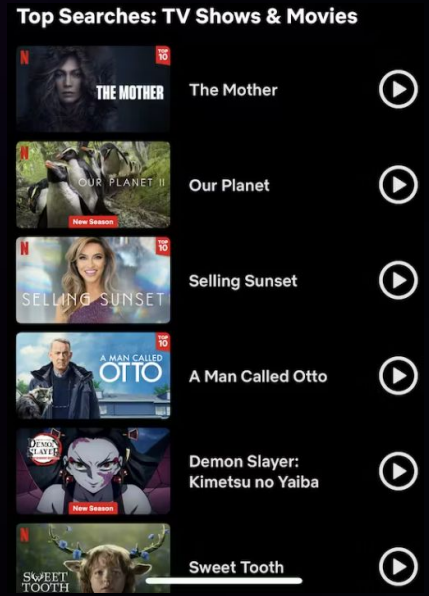
Section

Entity

SCHEMA DESIGN

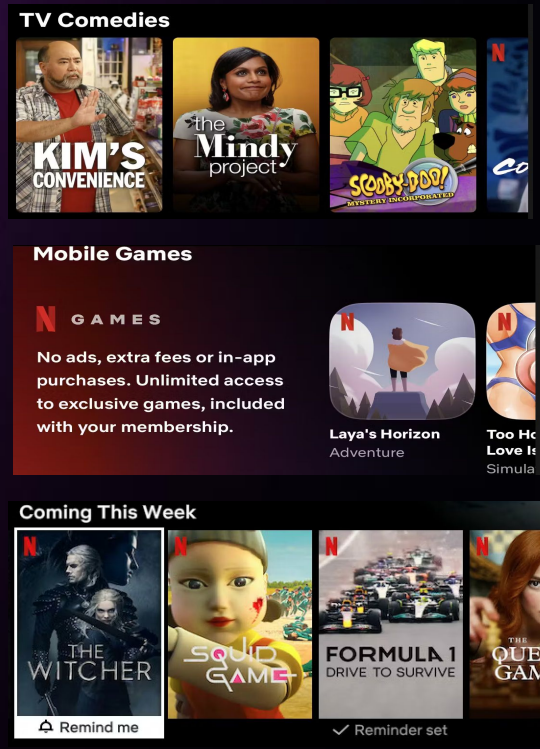
Section Types





Treatments: The Signal on Top

SAME SECTION (CAROUSEL) · DIFFERENT SECTION TREATMENTS



treatment=STANDARD

treatment=GAMES

treatment=COMING_SOON

SCHEMA DESIGN

Treatments: Same Entity, Different Signals

SAME ENTITY (VIDEO) · DIFFERENT ENTITY TREATMENTS

Boxshot

Portrait poster, title below

StandardBillboard

Wide hero art, synopsis, button

CharacterCircle

Character image inside a circle

RankedBoxshot

Boxshot + rank number overlay

SCHEMA DESIGN

Three Independent Axes

Section : Carousel

Section Treatment: Standard

Entity Treatment : Boxshot

Section: Carousel

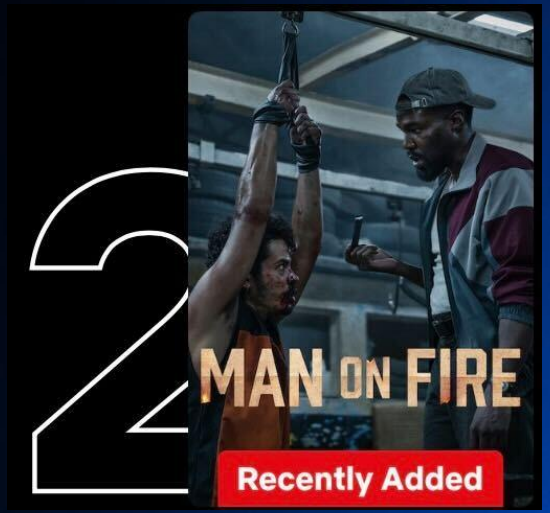
Section Treatment:Continue Watching

Entity Treatment: Boxshot

Section: Carousel

Section Treatment: Standard

Entity Treatment: → Ranked



SCHEMA DESIGN

The Schema in Practice

GRAPHQL SCHEMA

```
interface PageBase {
      id: ID!
      version: String!
      sections(first: Int): SectionConnection
}

interface BaseSection {
      id: ID!
      sectionTreatment: SectionTreatment
}

interface Entity {
      entityId: ID!
      entityTreatment: EntityTreatment
}

type BrowsePage implements PageBase { ... }
```

SAMPLE RESPONSE PAYLOAD

```
{
"page": {
        "id": "browse",
        "sections": {
        "edges": [{
        "node": {
                "__typename": "CarouselSection",
                "sectionTreatment": {
                "__typename": "BillboardSectionTreatment"
                },
                "items": [{
                        "entity":
                        {
                        "__typename": "BoxShotEntityTreatment",
                        "title": "Stranger Things"
                        }
                }]
                }
        }]
        }
        }
        }
```

SCHEMA DESIGN

Page Capabilities: The Contract

NOT THIS

API Versioning

Schema Introspection

THIS

Capabilities

Client declares what it can render. Server returns only that.
No versioning. No negotiation.

SCHEMA DESIGN

Declaring Page Capabilities

Every request carries a declaration of what the client can render. The server filters accordingly.

SCHEMA

```
input PageCapabilities {
   sectionTypes: [String!]!
   sectionTreatments: [String!]!
   entityTreatments: [String!]!
}

type Query {
   page(
   id: ID!
   capabilities: PageCapabilities!
  ): PageBase
}
```

CLIENT REQUEST

```
query GetPage {
page(
id: "browse"
capabilities: {
   sectionTypes: [ "CarouselSection" ]
   sectionTreatments: [
     "BillboardSectionTreatment"
     "StandardSectionTreatment"
   ]
  entityTreatments: [
" BoxshotEntityTreatment"
  "ContinueWatchingEntityTreatment"
]
}
) {
id
sections { ... }
}
}
```

SCHEMA DESIGN

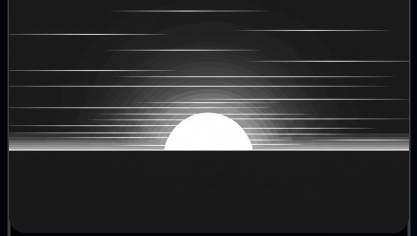
The Lifecycle of a Schema Type

`@experimental`

Opted-in clients only. Gated by capabilities.

`stable`

Renamed. Permanent. Ships to all clients.



~~`@deprecated`~~

Still returns data. Old clients remain safe.

`removed`

Zero active traffic. No breakage occurred.

one direction: forward

SCHEMA DESIGN

Moving behaviors to Server

ACT I: THE OLD WAY

Hardcoded Client Knowledge

ACT II: THE SDUI PROBLEM

The Knowledge Gap

ACT III: THE SOLUTION

Dynamic Instruction Set

SCHEMA DESIGN

Triggers: How the Server Hooks Into Client Events

Triggers

```
type Page {
id: ID!
sections: [Section]!
triggers: [Trigger] // New: Page-level hooks
}

type Section {
id: ID!
treatment: Treatment!
triggers: [Trigger] // Contextual to this UI area
}
```

TRIGGER TYPES + ACTIONS

```
union Trigger =
| PlaybackEndTrigger
| ServerNotificationTrigger
| AddToMyListTrigger

union Action =
| UpdatePageAction
| NewPageAction
| ApplyPrefetchAction

type PlaybackEndTrigger {
on: Event!
do: Action!
}
```

SCHEMA DESIGN

Pages That Update Themselves

The client does not decide which sections change. The server does.

01

Server returns page

triggers attached

02

User action fires

e.g. video exits

03

Device calls server

page update request

04

Server returns sections

updated content only

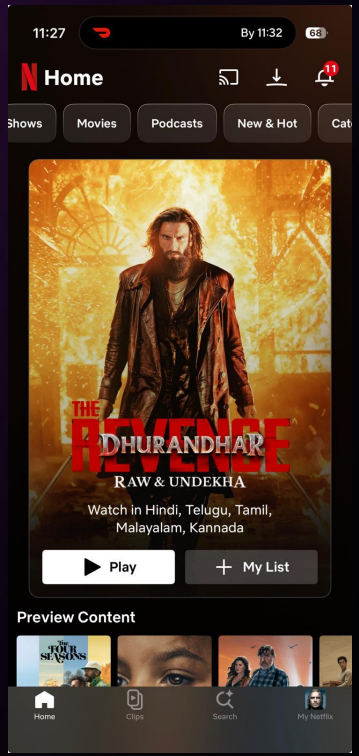
05

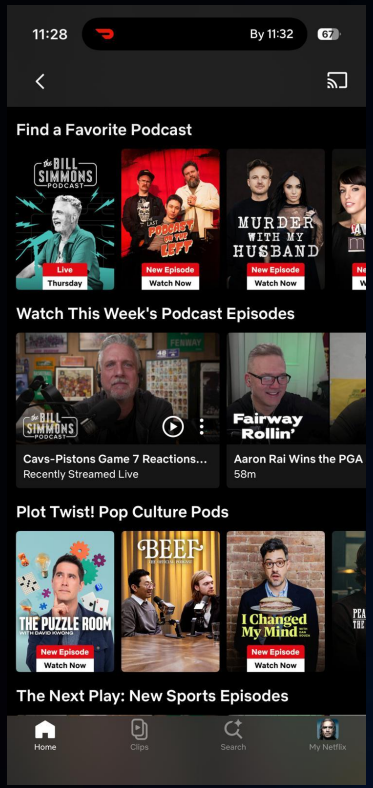
Client re-renders

no bespoke logic

SCHEMA DESIGN

Build a Page with Me

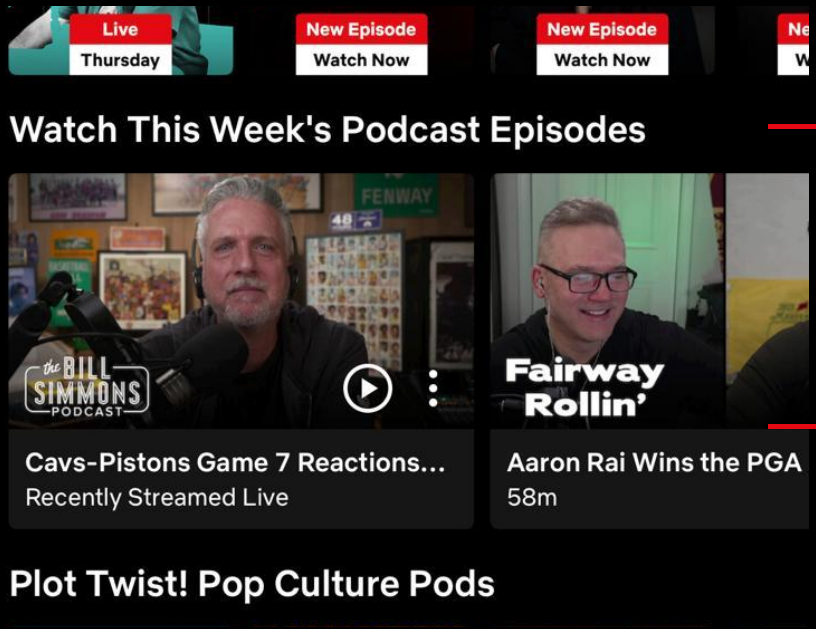




SCHEMA DESIGN

Build a Page with Me

THE PAGE



THE PAYLOAD

```
{
  "page": {
    "__typename": "PodcastPage",
    "id": "podcast",
    "sections": [
      {
        "__typename": "CarouselSection",
        "sectionTreatment": { "__typename":
        "EpisodicSectionTreatment" },
        "triggers": [
          {
            "__typename": "PlaybackEndTrigger",
            "action": {
              "__typename": "UpdateSectionAction"
            }
        }]
        "items": [{
          "entityId": "episode_001",
          "entityTreatment": {
          "__typename": "EpisodicEntityTreatment" }
        }]
      }
    ]
  }
}
```

SCHEMA DESIGN

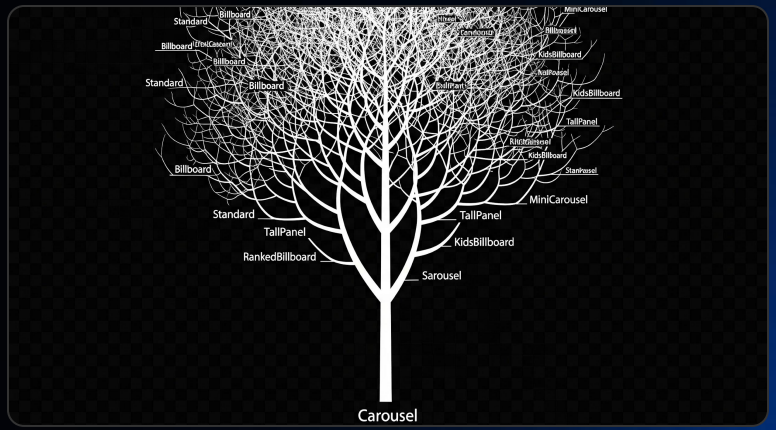
What Worked. What We Learned

WORKED

Ship in minutes. Not releases.

Features, layouts, and experiments go live the moment the server returns them. No app store. No coordination.

LEARNING



Schema should tie into design.

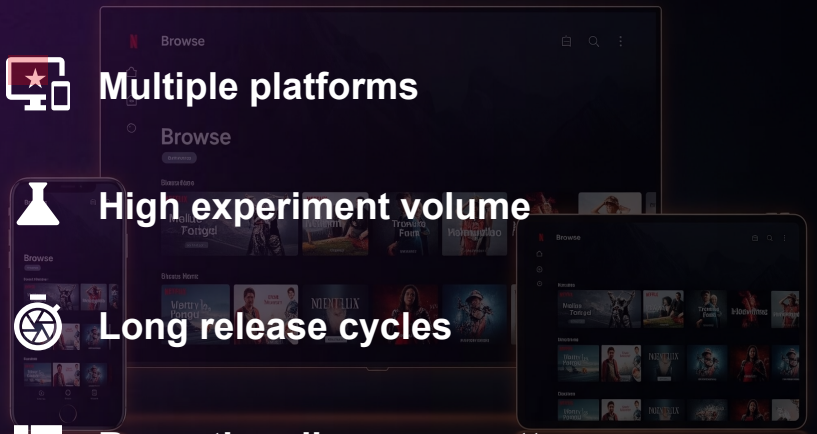
Every minor visual variation became a new treatment. Near-duplicates compounded. Schema and design drifted apart.

APPLICABILITY

Does This Pattern Fit Your Problem?

Fits Well

- Multiple platforms
- High experiment volume
- Long release cycles
- Repeating discovery patterns



Think Twice

- Highly stateful UI
- One-off surfaces
- Single platform
- Fine-grained layout control

The schema is the **product roadmap.**

Evolve it deliberately.


NETFLIX
Questions ?